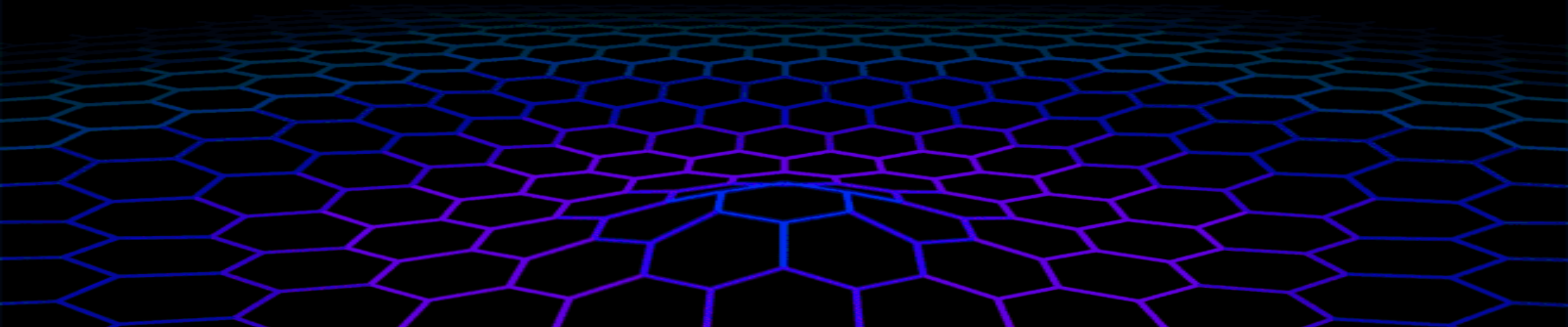
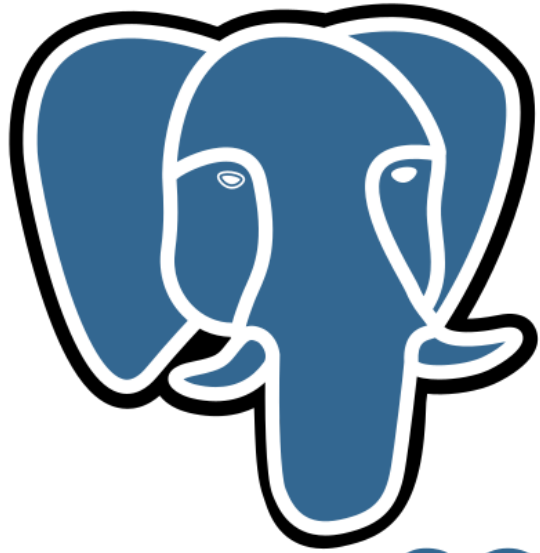


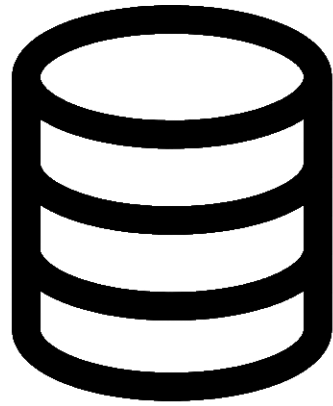
Banco de Dados I







PostgreSQL



SQL



Microsoft®
SQL Server®

ORACLE



MySQL™

Tools

MySQL - <https://dev.mysql.com/downloads>

MariaDB - <https://mariadb.org>

WampServer - <https://www.wampserver.com/en/download-wampserver-64bits>

Xampp - https://www.apachefriends.org/pt_br/index.html

Chocolatey - <https://chocolatey.org>

Atom - <https://atom.io>

Brackets - <http://brackets.io>

DataGrip - <https://www.jetbrains.com/datagrip>

HeidiSQL - <https://www.heidisql.com>

Notepad++ - <https://notepad-plus-plus.org/downloads>

SquirrelL - <http://www.squirrelsql.org>

Sublime Text – <https://www.sublimetext.com/download>

Visual Studio Code - <https://code.visualstudio.com>

SQL - Structured Query Language - Linguagem de Consulta Estruturada

DDL (Data Definition Language):

- **CREATE:** Cria novos objetos no banco de dados.

```
CREATE TABLE alunos (  
    id            INT,  
    nome          VARCHAR(100),  
    data_nascimento DATE,  
    PRIMARY KEY (id)  
);
```

- **ALTER:** Modifica a estrutura de um objeto existente.

```
ALTER TABLE alunos ADD COLUMN email VARCHAR(100);
```

- **DROP:** Remove objetos do banco de dados.

```
DROP TABLE alunos;
```

SQL - Structured Query Language - Linguagem de Consulta Estruturada

DML (Data Manipulation Language):

- **SELECT:** Busca dados no banco.

```
SELECT nome, data_nascimento FROM alunos WHERE id = 1;
```

- **INSERT:** Insere novos registros.

```
INSERT INTO alunos (id, nome, data_nascimento)  
VALUES (1, 'Eduarda Vulnária', '1995-06-15');
```

- **UPDATE:** Atualiza registros existentes.

```
UPDATE alunos  
SET nome = 'Pedro Gumercinco'  
WHERE id = 3;
```

- **DELETE:** Remove registros do banco.

```
DELETE FROM alunos WHERE id = 1;
```



O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Structured Query Language) como interface.

1970s: Surgem os primeiros SGBDs baseados em SQL.

1986: ANSI define o padrão SQL-86 (SQL 1).

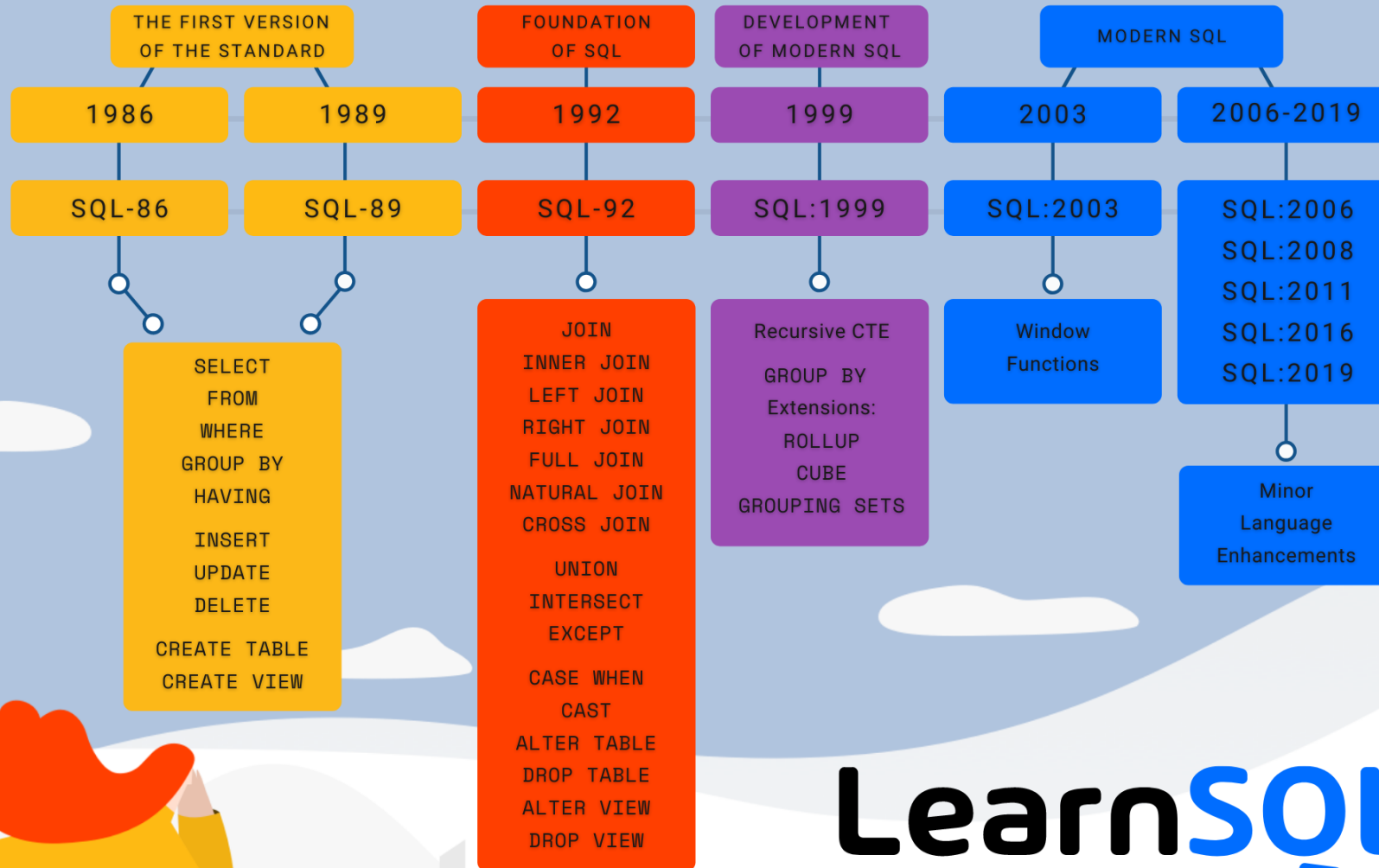
1989: SQL-89 (pequenas revisões).

1992: SQL-92 (versão mais robusta, amplamente utilizada).

1999: SQL:1999 (introduz suporte a triggers e procedimentos armazenados).

2003 - 2016: Atualizações periódicas no padrão SQL.

The History of SQL Standards



Instalando MySQL através do Chocolatey no Windows

Instalar o Chocolatey

Abra o PowerShell como administrador e execute o comando abaixo para instalar o Chocolatey:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; `[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

Instalar o MySQL com Chocolatey

Abra novamente o PowerShell como administrador e execute o comando:

```
choco install mysql
```

Inicie o serviço MySQL com o comando:

```
net start mysql
```

Se desejar atualizar o MySQL execute:

```
choco upgrade mysql
```

Trabalhando com MySQL via Console (Windows)

Abrindo o MySQL no Terminal:

1. Abra o **Prompt de Comando** (cmd) do Windows.
2. Execute o seguinte comando para acessar o MySQL:
3. **mysql -u root -p**
4. O terminal solicitará a senha. Insira a senha que você definiu durante a instalação do MySQL (ou **senacrs** nas máquinas da faculdade).

Trabalhando com MySQL via Console (Windows)

Comandos Básicos:

Listar Bancos de Dados:

SHOW DATABASES;


```
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
+-----+
```

Criar um Banco de Dados:

CREATE DATABASE exemplo;

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| exemplo |  
+-----+
```

Trabalhando com MySQL via Console (Windows)

 Administrador: Windows PowerShell

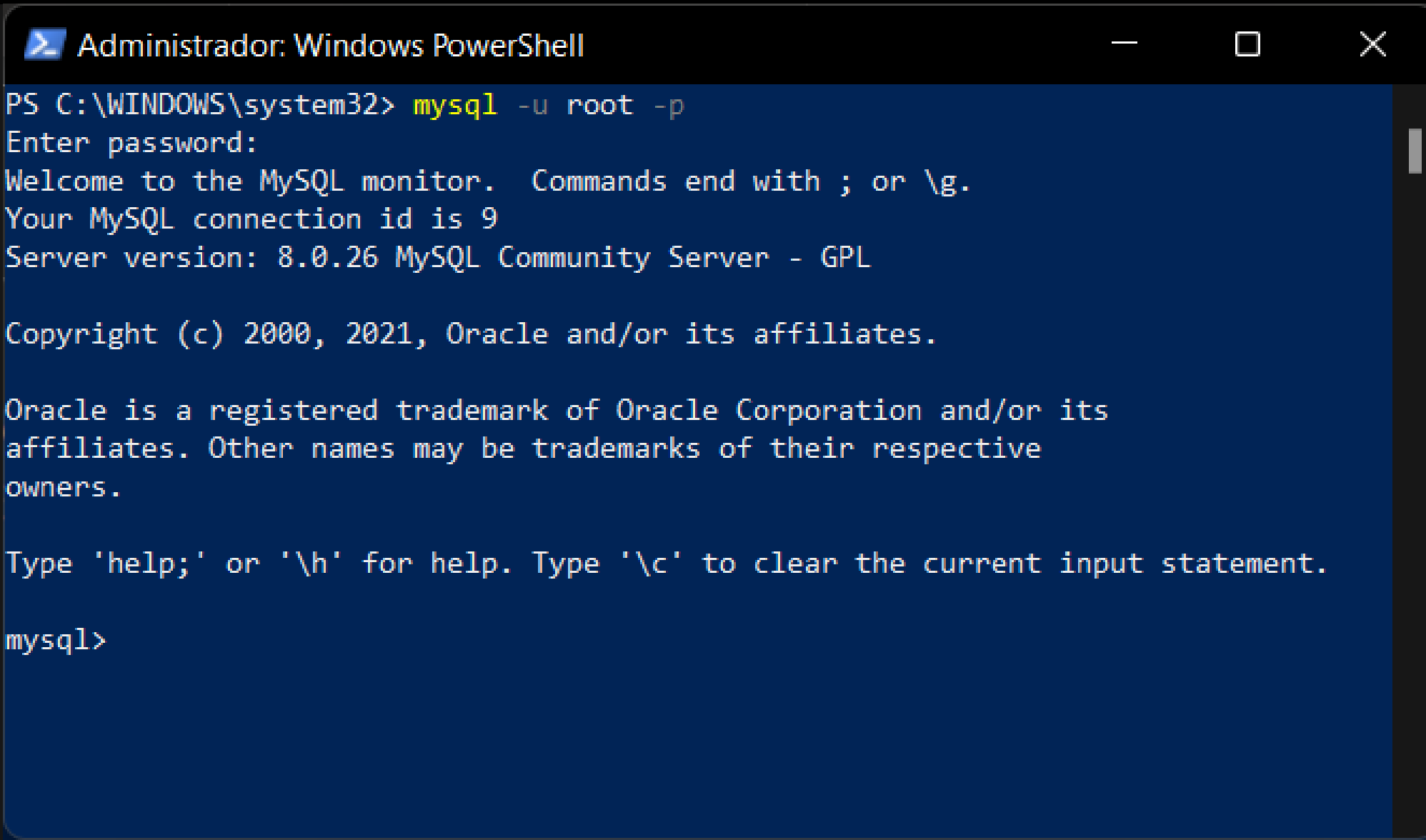
```
PS C:\WINDOWS\system32> mysql -u root -p
```

Trabalhando com MySQL via console

 Administrador: Windows PowerShell

```
PS C:\WINDOWS\system32> mysql -u root -p
Enter password: █
```

Trabalhando com MySQL via console



```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.26 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Databases - Nomenclatura

- Máximo de 64 caracteres.
- **Permitido**: letras, números, traços (-), underlines (_).

Exemplos: `meu_banco`, `dados2024`

- **Proibido**: barras (/), pontos (.), e acentuações.

Exemplos: `meu.banco`, `dados/2024`, `banco do país`

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax.
```

Dicas:

Evitar o uso de **acentos** e **cedilhas** para garantir compatibilidade em diferentes sistemas.

Utilize nomes descritivos e curtos para facilitar o entendimento e o gerenciamento dos bancos de dados, como `farmacia_db` ou `projeto_marketing`.

Utilizando a Cláusula IF NOT EXISTS para Criar Bancos de Dados

A cláusula **IF NOT EXISTS** é usada para evitar erros quando tentamos criar um banco de dados que já existe.

```
ERROR 1007 (HY000): Can't create database 'exemplo'; database exists
```

Exemplo: **CREATE DATABASE IF NOT EXISTS** exemplo;

Dica:

Utilizar **IF NOT EXISTS** é uma boa prática em scripts de inicialização que podem ser executados repetidamente, para evitar erros de duplicação.

Selecionando um banco de dados

Para trabalhar com um banco de dados específico no MySQL, primeiro você precisa selecioná-lo usando o comando **USE**

```
USE exemplo;
```

Dica:

Sempre verifique se o banco de dados foi selecionado corretamente antes de executar comandos SQL.

```
ERROR 1049 (42000): Unknown database 'exemplo'
```

Excluindo um Banco de Dados no MySQL

Para excluir um banco de dados e todos os dados nele contidos de forma permanente, utilize o seguinte comando:

```
DROP DATABASE exemplo;
```

Uso da Cláusula **IF EXISTS**:

Para evitar erros ao tentar excluir um banco de dados que não existe, utilize a cláusula IF EXISTS:

```
DROP DATABASE IF EXISTS exemplo;
```

Atenção: ⚠ Excluir um banco de dados é uma ação permanente. Todos os dados serão perdidos e não poderão ser recuperados.

Dica:

Boas práticas: Utilizar a cláusula **IF EXISTS** é útil em scripts automatizados, para garantir que a exclusão seja feita apenas se o banco de dados realmente existir, evitando erros desnecessários.

Criando Tabelas no MySQL

Para criar uma tabela no MySQL, usamos o comando **CREATE TABLE**, onde definimos os campos (colunas) e seus respectivos tipos de dados.

Exemplo básico:

```
CREATE TABLE aluno (  
    id      INT,  
    nome    VARCHAR(100),  
    email   VARCHAR(100),  
    PRIMARY KEY (id)  
);
```

Criando Tabelas no MySQL

Para criar uma tabela no MySQL, usamos o comando **CREATE TABLE**, onde definimos os campos (colunas) e seus respectivos tipos de dados.

Exemplo básico:

```
CREATE TABLE aluno (  
  
    id      INT,  
    -- Define o campo id como um número inteiro. Ele servirá como identificador único para cada registro.  
  
    nome    VARCHAR(100),  
    -- O campo nome pode armazenar até 100 caracteres de texto. Isso é ideal para nomes curtos e médios.  
  
    email   VARCHAR(100),  
    -- O campo email também pode armazenar até 100 caracteres, adequado para endereços de email.  
  
    PRIMARY KEY (id)  
    -- Define o campo id como chave primária, garantindo que cada valor em id seja único em toda a tabela.  
);
```

Criando Tabelas no MySQL

Dica para Chaves Compostas:

Se precisar criar uma chave composta, defina desta forma:

```
CREATE TABLE aluno (  
    id          INT,  
    nome        VARCHAR(100),  
    email       VARCHAR(100),  
    PRIMARY KEY (id, email) /* Define uma chave composta entre os campos id e  
email, garantindo que a combinação seja única. */  
);
```

Criando Tabelas no MySQL

```
DROP DATABASE IF EXISTS aula06;  
CREATE DATABASE IF NOT EXISTS aula06;  
USE aula06;
```

```
CREATE TABLE teste(  
  codigo INT(11),  
  nome VARCHAR(100),  
  UF CHAR(2)  
);
```

Nome da tabela

Nome dos campos

Tipos dos campos

Adicionando Restrições no MySQL (**NOT NULL** e **UNIQUE**)

```
CREATE TABLE alunos (  
    id          INT,  
    -- Define o campo id como um número inteiro.  
    nome        VARCHAR(150) NOT NULL,  
    -- O campo nome é obrigatório. Não pode ser nulo.  
    email       VARCHAR(150) UNIQUE,  
    -- O campo email deve ser único. Nenhum outro registro pode ter o mesmo email.  
    nascimento  DATE,  
    -- Define o campo nascimento para armazenar datas.  
    PRIMARY KEY (id)  
    -- Define o campo id como chave primária, garantindo sua unicidade.  
);
```

Dica:

O uso de restrições como **NOT NULL** e **UNIQUE** é fundamental para garantir a integridade dos dados e evitar valores duplicados ou em branco em campos importantes.

Criando Chaves Compostas no MySQL

Dica para Chaves Compostas:

Se precisar criar uma chave composta, defina desta forma:

```
CREATE TABLE aluguel (  
    placa_veiculo CHAR(7),  
    -- O campo placa_veiculo armazena a placa do veículo com exatamente 7 caracteres.  
    rg_locador CHAR(15),  
    -- O campo rg_locador armazena o RG do locador com até 15 caracteres.  
    data_hora DATETIME,  
    -- O campo data_hora armazena a data e hora da locação.  
    PRIMARY KEY (placa_veiculo, rg_locador, data_hora)  
    -- Define uma chave composta usando placa_veiculo, rg_locador e data_hora.  
);
```

Dica:

As chaves compostas são usadas quando nenhum campo isolado pode identificar de forma única os registros, mas a combinação de múltiplos campos pode garantir essa unicidade.

Auto Incremento no MySQL

```
CREATE TABLE produtos (  
    codigo    INT AUTO_INCREMENT,  
    -- O campo codigo será preenchido automaticamente, incrementando a cada novo registro.  
    nome      VARCHAR(100),  
    -- O campo nome pode armazenar até 100 caracteres.  
    preco     DECIMAL(10, 2),  
    -- O campo preco armazena valores numéricos com duas casas decimais.  
    PRIMARY KEY (codigo)  
    -- Define o campo codigo como chave primária.  
);
```

Dica:

O uso de **AUTO_INCREMENT** é ideal quando queremos que o campo de identificação seja preenchido automaticamente com valores únicos, enquanto **NOT NULL** garante que o campo seja obrigatório.

Praticando

```
CREATE TABLE clientes (  
    codigo      INT AUTO_INCREMENT,  
    -- O campo codigo será preenchido automaticamente, incrementando com cada novo registro.  
    nome        VARCHAR(100) NOT NULL,  
    -- O campo nome é obrigatório e pode armazenar até 100 caracteres.  
    email       VARCHAR(100),  
    -- O campo email pode armazenar até 100 caracteres.  
    telefone    CHAR(20),  
    -- O campo telefone tem exatamente 20 caracteres.  
    cidade      VARCHAR(100),  
    -- O campo cidade pode armazenar até 100 caracteres.  
    PRIMARY KEY (codigo)  
    -- Define o campo codigo como chave primária.  
);
```

Dica:

O **AUTO_INCREMENT** é usado frequentemente em colunas de chave primária, garantindo que cada novo registro receba um identificador único automaticamente.

Visualizando Tabelas no MySQL

Comando para listar tabelas:

```
SHOW TABLES;  
-- Exibe todas as tabelas existentes no banco de dados atual.
```

Comando para ver a estrutura de uma tabela:

```
DESCRIBE clientes;  
/* Exibe a estrutura da tabela 'clientes', incluindo nome das colunas, tipo de  
dados e restrições. */
```

Dica:

Use o comando **DESCRIBE** sempre que precisar verificar a estrutura de uma tabela e entender quais colunas e restrições foram aplicadas

Inserindo Dados em Tabelas

```
INSERT INTO clientes (nome, email, cidade)
VALUES ('Clarinda Nunes', 'clarinda.nunes@email.com', '567-1234', 'Pelotas');
-- Insere um novo registro na tabela clientes.
```

INSERT INTO clientes: Especifica que estamos inserindo dados na tabela clientes.

(nome, email, telefone, cidade): Define quais campos receberão os dados. Não precisamos inserir o campo código porque ele será preenchido automaticamente.

VALUES ('Clarinda Nunes', 'clarinda.nunes@email.com', '567-1234', 'Pelotas'): Fornece os valores para nome, email e telefone para o novo cliente.

Dica:

Ao utilizar **AUTO_INCREMENT**, não é necessário inserir o valor para a chave primária, pois ela será gerada automaticamente.

Inserindo Múltiplos Registros com INSERT

```
INSERT INTO clientes (nome, email, telefone)
VALUES
    ('Maria Silva', 'maria.silva@email.com', '567-1234', 'Pelotas'),
    ('João Santos', 'joao.santos@email.com', '567-5678', 'Chuí'),
    ('Bruce Wayne', 'bruce.wayne@email.com', '567-5678', 'Arroio Grande'),
    ('Clark Kent', 'clark.kent@email.com', '567-5678', 'Porto Alegre'),
    ('Peter Parker', 'peter.parker@email.com', '567-5678', 'Pelotas'),
    ('Diana Prince', 'diana.prince@email.com', '567-5678', 'Jaguarão'),
    ('Tony Stark', 'tony.stark@email.com', '567-5678', 'Herval'),
    ('Barry Allen', 'barry.allen@email.com', '567-5678', 'Morro Redondo'),
    ('Hal Jordan', 'hal.jordan@email.com', '567-5678', 'Capão do Leão'),
    ('Fred Flintstone', 'fred.flintstone@email.com', '567-5678', 'Chuí'),
    ('Bruce Banner', 'bruce.banner@email.com', '567-5678', 'Pelotas'),
    ('Ana Costa', 'ana.costa@email.com', '567-9876', 'Arroio Grande');
-- Insere doze registros na tabela clientes.
```

Dica:

Inserir múltiplos registros em uma única consulta economiza tempo e recursos do banco de dados, tornando o processo de inserção mais eficiente.

Inserindo Múltiplos Registros com INSERT

```
INSERT INTO produtos (nome, preco)
VALUES
    ('Notebook Acer Aspire 5', 3500.00),
    ('Mouse Logitech MX Master 3', NULL),
    ('Monitor Samsung 24"', 899.99),
    ('Teclado Mecânico Razer BlackWidow', 600.00),
    ('Fone de Ouvido Sony WH-1000XM4', 1500.00),
    ('SSD Kingston A400 480GB', 350.00),
    ('Smartphone Samsung Galaxy S21', NULL),
    (NULL, 1200.00),
    ('Impressora Multifuncional HP DeskJet', 500.00),
    ('Placa de Vídeo NVIDIA RTX 3070', 4500.00);
```

Recuperando Dados com SELECT

Comando para recuperar todos os registros:

```
SELECT * FROM clientes;  
-- Recupera todos os registros e colunas da tabela 'clientes'.
```

Comando para recuperar colunas específicas:

```
SELECT nome, email FROM clientes;  
-- Recupera apenas os campos 'nome' e 'email' de todos os registros.
```

Dica:

Sempre que possível, especifique as colunas que deseja recuperar ao invés de usar *. Isso melhora a performance e evita trazer dados desnecessários.

Limitação de Resultados com LIMIT

Comando para Limitar Resultados:

```
SELECT * FROM clientes  
LIMIT 5;  
-- Retorna apenas os primeiros 5 registros da tabela clientes.
```

LIMIT 5: Restringe o número de registros retornados para 5, exibindo apenas os primeiros 5 registros da tabela.

Comando com OFFSET:

```
SELECT * FROM clientes  
LIMIT 5 OFFSET 10;  
-- Retorna 5 registros, começando a partir do 11º (usando OFFSET 10).
```

OFFSET 10: Pula os primeiros 10 registros e começa a exibir a partir do 11º. Usado em conjunto com **LIMIT** para paginação.

Dica:

O uso de **LIMIT** e **OFFSET** é útil para paginar resultados, especialmente em aplicações que lidam com grandes volumes de dados.

Ordenando Resultados com ORDER BY

Comando para Atualizar Múltiplos Registros:

```
SELECT * FROM clientes  
ORDER BY nome;  
-- Ordena os registros de clientes em ordem alfabética crescente pelo campo nome
```

ORDER BY nome: Ordena os resultados pela coluna nome em ordem alfabética crescente (A-Z).

Comando para Ordenar em Ordem Decrescente:

```
SELECT * FROM clientes  
ORDER BY nome DESC;  
-- Ordena os registros de clientes em ordem alfabética decrescente (Z-A).
```

Dica:

ORDER BY pode ser usado em qualquer coluna da tabela. Além disso, combinações como **ORDER BY** nome, email permitem ordenar por mais de um critério.

Filtrando Resultados com WHERE

Comando para Filtrar Registros:

```
SELECT * FROM clientes
WHERE cidade = 'Pelotas';
-- Retorna apenas os clientes que moram em Pelotas.
```

WHERE cidade = 'Pelotas': Filtra os resultados para exibir apenas os registros onde a coluna cidade tem o valor 'Pelotas'.

Comando com Vários Critérios:

```
SELECT * FROM clientes
WHERE cidade = 'Pelotas' AND telefone IS NOT NULL;
-- Retorna clientes de Pelotas que têm um número de telefone cadastrado.
```

AND telefone **IS NOT NULL**: Combina duas condições. O comando filtra registros onde a cidade é 'Pelotas' e o campo telefone não é nulo.

Dica:

O operador **AND** permite combinar múltiplas condições, enquanto o **OR** retorna registros que atendem a pelo menos uma das condições especificadas.

Usando Funções de Agregação no MySQL

Comando para Contar Registros:

```
SELECT COUNT(*) FROM clientes;  
-- Conta o número total de registros na tabela clientes.
```

`COUNT(*)`: Conta todos os registros da tabela, independentemente dos valores nas colunas.

Comando para Calcular Média:

```
SELECT AVG(preco) FROM produtos;  
-- Calcula a média dos preços dos produtos.
```

`AVG(preco)`: Calcula a média dos valores na coluna preco.

Dica:

Funções de agregação como **SUM**, **MAX**, **MIN** e **AVG** são úteis para gerar relatórios e análises estatísticas.

Atualizando Dados no MySQL com UPDATE

Comando para Atualizar Registros:

```
UPDATE clientes  
SET email = 'novo.email@email.com'  
WHERE codigo = 1;  
-- Atualiza o campo email do cliente com código 1.
```

- **UPDATE** clientes: Especifica que estamos atualizando dados na tabela clientes.
- **SET** email = 'novo.email@email.com': Define que o campo email será alterado para o novo valor especificado.
- **WHERE** codigo = 1: Garante que a atualização ocorra apenas no registro onde o codigo é igual a 1.

Dica:

Sempre use uma cláusula **WHERE** ao atualizar dados, para evitar modificar todos os registros da tabela por acidente.

Atualizando Dados no MySQL com UPDATE

Comando para Atualizar Múltiplos Registros:

```
UPDATE clientes  
SET telefone = '567-0000'  
WHERE cidade = 'Pelotas';  
-- Atualiza o campo telefone de todos os clientes da cidade de Pelotas.
```

`WHERE cidade = 'Pelotas':` Atualiza o campo telefone para todos os clientes que moram em Pelotas

Excluindo Dados no MySQL com DELETE

Comando para Excluir Registros:

```
DELETE FROM clientes  
WHERE codigo = 1;  
-- Remove o cliente com código 1 da tabela clientes.
```

Comando para Excluir Múltiplos Registros:

```
DELETE FROM clientes  
WHERE cidade = 'Pelotas';  
-- Remove todos os clientes que moram na cidade de Pelotas.
```

`WHERE cidade = 'Pelotas':` Remove todos os registros de clientes que moram em Pelotas.

Dica:

Sempre tenha **cuidado** ao usar o comando **DELETE** e a cláusula **WHERE**, pois deletar registros é uma ação irreversível.

Tipos de Dados Numéricos no MySQL

Inteiros:

INT: Número inteiro com 4 bytes. Intervalo: -2.147.483.648 a 2.147.483.647.

TINYINT: Número inteiro pequeno com 1 byte. Intervalo: -128 a 127.

SMALLINT: Número inteiro com 2 bytes. Intervalo: -32.768 a 32.767.

MEDIUMINT: Número inteiro com 3 bytes. Intervalo: -8.388.608 a 8.388.607.

BIGINT: Número inteiro grande com 8 bytes. Intervalo: -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Tipos de Dados Numéricos no MySQL

Decimal:

DECIMAL(M, D): Números decimais exatos com M dígitos e D casas decimais.

FLOAT: Números de ponto flutuante com precisão simples (4 bytes).

DOUBLE: Números de ponto flutuante com precisão dupla (8 bytes).

Dica:

Use **INT** para contadores e **DECIMAL** para valores monetários, onde a precisão é importante.

Tipos de Dados de Data e Hora no MySQL

Data:

DATE: Armazena datas no formato 'AAAA-MM-DD'. Exemplo: '2024-01-01'.

DATETIME: Armazena data e hora combinadas. Exemplo: '2024-01-01 12:34:56'.

TIMESTAMP: Armazena data e hora com base no UTC. Útil para rastrear alterações em fusos horários diferentes.

Hora:

TIME: Armazena um valor de hora no formato 'HH:MM'. Exemplo: '12:34:56'.

YEAR: Armazena apenas o ano em formato de 4 dígitos. Exemplo: '2024'.

Dica:

Use **TIMESTAMP** para registros de data e hora que podem variar com o fuso horário, e **DATETIME** para valores estáticos de data e hora.

Tipos de Dados de Texto no MySQL

Texto Curto:

CHAR(N): Armazena cadeias de caracteres de comprimento fixo, até N caracteres. Exemplo: **CHAR(5)**.

VARCHAR(N): Armazena cadeias de caracteres de comprimento variável, até N caracteres. Exemplo: **VARCHAR(255)**.

Texto Longo:

TEXT: Armazena até 65.535 caracteres.

MEDIUMTEXT: Armazena até 16.777.215 caracteres.

LONGTEXT: Armazena até 4.294.967.295 caracteres.

Dica:

Use **VARCHAR** para textos curtos e **TEXT** para grandes blocos de texto como descrições.

Tipos de Dados Binários no MySQL

Binários:

BINARY(N): Semelhante a CHAR, mas armazena dados binários de comprimento fixo.

VARBINARY(N): Semelhante a VARCHAR, mas armazena dados binários de comprimento variável.

Exemplo: **BINARY(16)** armazena exatamente 16 bytes de dados. **BINARY(8)** armazena 8 bytes de dados.

Binários Longos:

BLOB: Armazena dados binários até 65.535 bytes.

MEDIUMBLOB: Armazena dados binários até 16.777.215 bytes.

LONGBLOB: Armazena dados binários até 4.294.967.295 bytes.

Dicas:

Use **BINARY** quando você precisa garantir que todos os valores armazenados tenham o mesmo tamanho.

Use **VARBINARY** quando os dados binários podem variar em tamanho, para otimizar o uso de espaço.

Use **BLOB** para armazenar arquivos binários como imagens ou documentos.

SET

SET

Permite que o usuário faça uma escolha dado determinado número de opções. Cada campo pode conter até, 64 opções.

Exemplo:

```
curso SET('informatica', 'geomatica') NOT NULL;
```

Neste exemplo este campo pode conter apenas os seguintes itens:

'informatica'

'geomatica'

'informatica, geomatica'

ENUM

ENUM

Indicado para:

- conjunto de opções fixas
- menu *dropdown* de formulário

Semelhante ao SET, com a diferença que apenas um valor pode ser escolhido.

Exemplo:

```
turno ENUM('manhã', 'tarde', 'noite') NOT NULL;
```

Neste exemplo este campo pode conter os seguintes valores:

- 'manhã'
- 'tarde'
- 'noite'

SET e ENUM

Diferença entre o *datatype* **ENUM** e o *datatype* **SET**:

ENUM pode ter um (e apenas 1) valor escolhido de uma lista de opções. Esta lista pode ter até 65535 elementos.

SET pode ter zero a "n" valores escolhidos da lista de opções. Esta lista apenas pode ter 64 elementos.

Vamos praticar com o tio Glad

Preparando o terreno

```
CREATE TABLE clientes(  
    codigo    CHAR(6),  
    nome      VARCHAR(20),  
    dataN     DATE,  
    valor     DECIMAL(10,2),  
    cidade    VARCHAR(20),  
    PRIMARY KEY(codigo)  
);  
  
-- Posso fazer comentário em uma linha  
  
/* Posso fazer comentário  
em mais  
de uma linha */
```


Exemplo

- -- Inserindo registros na tabela clientes:

```
INSERT INTO clientes (codigo, nome, dataN, valor, cidade) VALUES ('B5200X', 'Bartolomeu', '1982-12-23', 100.4, 'Pelotas');
INSERT INTO clientes (codigo, nome, dataN, valor, cidade) VALUES ('A73111', 'Orpiliano', '2001-07-19', 98.6, 'Canoas');
INSERT INTO clientes (codigo, nome, dataN, valor, cidade) VALUES ('A77222', 'Dorvalinau', '1999-11-21', 900, 'Pelotas');
INSERT INTO clientes (codigo, nome, dataN, valor, cidade) VALUES ('B79321', 'Bartolomeu', '1969-01-24', 31.29, 'Erechim');
INSERT INTO clientes VALUES ('BX9843', 'Josivaldo', '1967-04-01', 93.45, 'Blumenau');
INSERT INTO clientes (codigo, nome) VALUES ('S9983W', 'Sarafina');
INSERT INTO clientes (cidade, nome, valor, codigo) VALUES ('Pelotas', 'Italinea', 1200.49, 'EL171C');
INSERT INTO clientes VALUES ('XYZ432', 'Anacleto', '1967-04-02', 90, 'Pelotas');
```

```
-- Para visualizar os registros da tabela clientes:
SELECT * FROM clientes;
```

Exemplo

Recuperar todas as colunas da tabela **clientes**.

Recuperar apenas as colunas **nome** e **valor** da tabela **clientes**.

Exemplo

Recuperar todas as colunas da tabela **clientes**.

```
SELECT * FROM clientes;
```

Recuperar apenas as colunas **nome** e **valor** da tabela **clientes**.

Exemplo

Recuperar todas as colunas da tabela **clientes**.

```
SELECT * FROM clientes;
```

Recuperar apenas as colunas **nome** e **valor** da tabela **clientes**.

```
SELECT nome, valor FROM clientes;
```

Exemplo

Recuperar apenas os registros de clientes da cidade **Pelotas**

Recuperar apenas os registros de clientes da cidade **Pelotas** com valor menor do que 100

Recuperar apenas os registros de clientes da cidade **Pelotas** ou da cidade **Erechim**

Recuperar apenas clientes com valor **acima** de **100**.

Recuperar apenas clientes que **não** são de **Pelotas**

Exemplo

Recuperar apenas os registros de clientes da cidade **Pelotas**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas';
```

Recuperar apenas os registros de clientes da cidade **Pelotas** com valor menor do que 100

Recuperar apenas os registros de clientes da cidade **Pelotas** ou da cidade **Erechim**

Recuperar apenas clientes com valor **acima** de **100**.

Recuperar apenas clientes que **não** são de **Pelotas**

Exemplo

Recuperar apenas os registros de clientes da cidade **Pelotas**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas';
```

Recuperar apenas os registros de clientes da cidade **Pelotas** com valor menor do que 100

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' AND valor < 100;
```

Recuperar apenas os registros de clientes da cidade **Pelotas** ou da cidade **Erechim**

Recuperar apenas clientes com valor **acima** de **100**.

Recuperar apenas clientes que **não** são de **Pelotas**

Exemplo

Recuperar apenas os registros de clientes da cidade **Pelotas**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas';
```

Recuperar apenas os registros de clientes da cidade **Pelotas** com valor menor do que 100

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' AND valor < 100;
```

Recuperar apenas os registros de clientes da cidade **Pelotas** ou da cidade **Erechim**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' OR cidade = 'Erechim';
```

Recuperar apenas clientes com valor **acima** de **100**.

Recuperar apenas clientes que **não** são de **Pelotas**

Exemplo

Recuperar apenas os registros de clientes da cidade **Pelotas**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas';
```

Recuperar apenas os registros de clientes da cidade **Pelotas** com valor menor do que 100

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' AND valor < 100;
```

Recuperar apenas os registros de clientes da cidade **Pelotas** ou da cidade **Erechim**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' OR cidade = 'Erechim';
```

Recuperar apenas clientes com valor **acima** de 100.

```
SELECT * FROM clientes WHERE valor > 100;
```

Recuperar apenas clientes que **não** são de **Pelotas**

Exemplo

Recuperar apenas os registros de clientes da cidade **Pelotas**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas';
```

Recuperar apenas os registros de clientes da cidade **Pelotas** com valor menor do que 100

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' AND valor < 100;
```

Recuperar apenas os registros de clientes da cidade **Pelotas** ou da cidade **Erechim**

```
SELECT * FROM clientes WHERE cidade = 'Pelotas' OR cidade = 'Erechim';
```

Recuperar apenas clientes com valor **acima** de 100.

```
SELECT * FROM clientes WHERE valor > 100;
```

Recuperar apenas clientes que **não** são de **Pelotas**

```
SELECT * FROM clientes WHERE cidade != 'Pelotas';
```

Exemplo

Alterar o nome do cliente **Bartolomeu** para **Bartoloteu**

Alterar a cidade do cliente de código 'A77222' para Satolep

Alterar para 90 o valor dos clientes da cidade 'Pelotas' que estejam com valor acima de 1000.

Alterar a dataN de todos os clientes para 1º de abril de 2000

Exemplo

Alterar o nome do cliente **Bartolomeu** para **Bartoloteu**

```
UPDATE clientes  
  SET nome = 'Bartoloteu'  
 WHERE nome = 'Bartolomeu';
```

Alterar a cidade do cliente de código 'A77222' para Satolep

Alterar para 90 o valor dos clientes da cidade 'Pelotas' que estejam com valor acima de 1000.

Alterar a dataN de todos os clientes para 1º de abril de 2000

Exemplo

Alterar o nome do cliente **Bartolomeu** para **Bartoloteu**

```
UPDATE clientes  
  SET nome = 'Bartoloteu'  
 WHERE nome = 'Bartolomeu';
```

Alterar a cidade do cliente de código 'A77222' para Satolep

```
UPDATE clientes  
  SET cidade = 'Satolep'  
 WHERE codigo = 'A77222';
```

Alterar para 90 o valor dos clientes da cidade 'Pelotas' que estejam com valor acima de 1000.

Alterar a dataN de todos os clientes para 1º de abril de 2000

Exemplo

Alterar o nome do cliente **Bartolomeu** para **Bartoloteu**

```
UPDATE clientes  
  SET nome = 'Bartoloteu'  
 WHERE nome = 'Bartolomeu';
```

Alterar a cidade do cliente de código 'A77222' para Satolep

```
UPDATE clientes  
  SET cidade = 'Satolep'  
 WHERE codigo = 'A77222';
```

Alterar para 90 o valor dos clientes da cidade 'Pelotas' que estejam com valor acima de 1000.

```
UPDATE clientes  
  SET valor = 90  
 WHERE cidade = 'Pelotas' AND valor > 1000;
```

Alterar a dataN de todos os clientes para 1º de abril de 2000

Exemplo

Alterar o nome do cliente **Bartolomeu** para **Bartoloteu**

```
UPDATE clientes  
  SET nome = 'Bartoloteu'  
 WHERE nome = 'Bartolomeu';
```

Alterar a cidade do cliente de código 'A77222' para Satolep

```
UPDATE clientes  
  SET cidade = 'Satolep'  
 WHERE codigo = 'A77222';
```

Alterar para 90 o valor dos clientes da cidade 'Pelotas' que estejam com valor acima de 1000.

```
UPDATE clientes  
  SET valor = 90  
 WHERE cidade = 'Pelotas' AND valor > 1000;
```

Alterar a dataN de todos os clientes para 1º de abril de 2000

```
UPDATE clientes  
  SET dataN = '2000-04-01';
```

Exemplo

Excluir apenas o cliente que tenha o código 'A73111'.

Excluir todos os registros de clientes.

Exemplo

Excluir apenas o cliente que tenha o código 'A73111'.

```
DELETE FROM clientes  
WHERE codigo = 'A73111';
```

Excluir todos os registros de clientes.

Exemplo

Excluir apenas o cliente que tenha o código 'A73111'.

```
DELETE FROM clientes  
WHERE codigo = 'A73111';
```

Excluir todos os registros de clientes.

```
DELETE FROM clientes;
```

Vamos praticar
novamente!

ATIVIDADE - Crie a modelagem lógica e física:

Criar um banco chamado **aula06**

Criar as tabelas:

EMPREGADO (matricula, nome, endereco, salario, matSuperv, codDep)

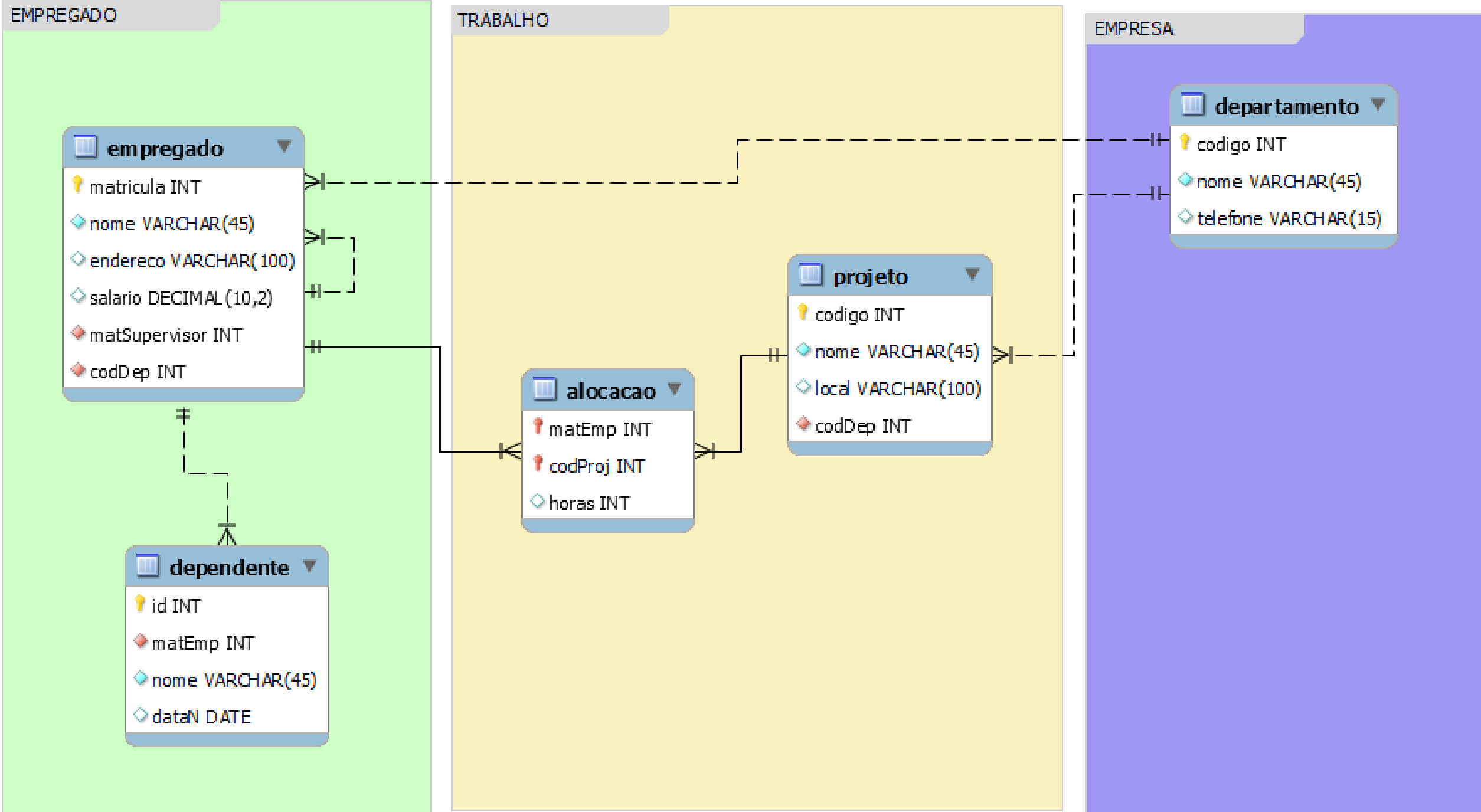
DEPARTAMENTO (codigo, nome, telefone)

PROJETO (codigo, nome, local, codDep)

ALOCACAO (matEmp, codProj, horas)

DEPENDENTE (id, matEmp, nome, dataN)

Solução



```
DROP DATABASE IF EXISTS aula06;  
CREATE DATABASE IF NOT EXISTS aula06;  
USE aula06;
```

```
CREATE TABLE departamento (  
    codigo    INT(11),  
    nome      VARCHAR(45) NOT NULL,  
    telefone  VARCHAR(15),  
    PRIMARY KEY (codigo)  
);
```

```
CREATE TABLE empregado (  
    matricula      INT(11),  
    nome           VARCHAR(45) NOT NULL,  
    endereco       VARCHAR(100),  
    salario        DECIMAL(10, 2),  
    matsupervisor  INT(11),  
    coddep         INT(11) NOT NULL,  
    PRIMARY KEY (matricula),  
    FOREIGN KEY (coddep) REFERENCES departamento (codigo),  
    FOREIGN KEY (matsupervisor) REFERENCES empregado (matricula)  
);
```



```
CREATE TABLE projeto(  
    codigo    INT(11),  
    nome      VARCHAR(45) NOT NULL,  
    local     VARCHAR(100),  
    coddep    INT(11) NOT NULL,  
    PRIMARY KEY (codigo),  
    FOREIGN KEY (coddep) REFERENCES departamento (codigo)  
);
```

```
CREATE TABLE alocacao(  
    matemp    INT(11),  
    codproj   INT(11),  
    horas     INT(11),  
    PRIMARY KEY (matemp, codproj),  
    FOREIGN KEY (matemp) REFERENCES empregado (matricula),  
    FOREIGN KEY (codproj) REFERENCES projeto (codigo)  
);
```

```
CREATE TABLE dependente (  
    id      INT(11),  
    matemp  INT(11),  
    nome    VARCHAR(45) NOT NULL,  
    datan   DATE,  
    PRIMARY KEY (id),  
    FOREIGN KEY (matemp) REFERENCES empregado (matricula)  
);
```

O que é uma Foreign Key no MySQL?

Definição:

Uma Foreign Key (chave estrangeira) é uma coluna (ou conjunto de colunas) que cria uma relação entre duas tabelas.

Ela refere-se à chave primária de outra tabela, estabelecendo uma ligação entre os dados dessas tabelas.

Finalidade:

Garantir a integridade referencial: A Foreign Key assegura que os dados em uma tabela (tabela filha) estejam sempre relacionados aos dados de outra tabela (tabela pai).

Prevenir dados órfãos: Impede que registros sejam inseridos ou atualizados com valores de chaves estrangeiras que não existam na tabela referenciada.

O que é uma Foreign Key no MySQL?

Vantagens do Uso de Foreign Keys:

Integridade de dados: Garante que um registro de uma tabela (como empregado) não possa existir sem um registro correspondente em outra tabela (como departamento).

Evita inconsistências: Se você tentar deletar um departamento que tenha empregados associados, o banco de dados pode bloquear a operação ou automaticamente deletar os registros relacionados, dependendo das opções de restrição definidas.

Foreign Key – Exemplo prático

No caso da tabela empregado, a coluna coddep é uma Foreign Key que referencia a coluna codigo da tabela departamento:

```
FOREIGN KEY (codproj) REFERENCES projeto (codigo)
ON DELETE RESTRICT
ON UPDATE CASCADE;
```

Sintaxe para Criar uma Foreign Key:

```
FOREIGN KEY (coluna_da_tabela_filha)
REFERENCES tabela_pai (coluna_da_tabela_pai)
```

Dica:

Integridade referencial pode ser configurada para impedir a exclusão de registros referenciados (com ON DELETE RESTRICT) ou permitir a exclusão em cascata (com ON DELETE CASCADE), o que também apaga os registros relacionados.

POPULANDO

- Cadastre empregados com salário menor do que 5.000, igual a 5.000 e maior do que 5.000;
- Cadastre pelo menos três funcionários com salário entre 1.700 e 2.800;
- Cadastre pelo menos 10 departamentos (com códigos de 1 até 10);
- Cadastre dois funcionários no departamento 1;
- Cadastre pelo menos três funcionários no departamento 2;
- Cadastre 3 funcionários no departamento 5;
- Cadastre alguns dependentes com data de nascimento igual a 27/10/2002;
- Cadastre alguns dependentes com data de nascimento posterior a 27/10/2002;

Sugestão de conteúdo para a tabela **departamento**.

codigo	nome	telefone
1	RH	321
2	Compras	123
3	Transportes	456
4	Marketing	654
5	Vendas	789
6	Financeiro	987
7	Estoque	234
8	Saúde	432
9	Controladoria Geral	345
10	Ouvidoria	543

```
INSERT INTO departamento (codigo, nome, telefone) VALUES (1, "RH", "321");
```

```
INSERT INTO departamento (codigo, nome, telefone) VALUES (2, "Compras", "123");
```

Sugestão de conteúdo para a tabela empregado.

matricula	nome	endereco	salario	mat supervisor	coddep
800	Josivaldo Antunes Nunes	Rua das flores	4000.00	NULL	1
835	Plinio Cabresto Selvagem	Av Sallus	5000.00	900	3
836	Ortega Raimundo Gomes	Av Marlua	5600.00	900	3
837	Solange Costa Ortiz	Rua Zanzibar	8000.00	835	3
841	Monange Costa Ortiz	Rua Zanzibar	1700.00	NULL	4
842	Rosange Costa Ortiz	Rua Zanzibar	1750.00	NULL	2
843	Violange Costa Ortiz	Rua Zanzibar	2700.00	NULL	5
844	Sustange Costa Ortiz	Rua Zanzibar	2800.00	843	6
845	Zuleiva Maciel Souza	Rua Troll	8050.00	NULL	1
846	Panceta Furunculo Anacleto	Rua Dolores	18050.00	844	1
847	Marciano das Antenas Verdes	Rua Marte	1050.00	NULL	2
848	Etevaldo Augusto Moraes	Rua Venus	2050.00	NULL	2
849	Lucrecio Borges Almeida	Rua Jupiter	8049.00	847	2
900	Marzivaniana Alves Breda	Rua das urtigas	4500.00	NULL	3
947	Zorzicleto Bicicleteiro	Rua Marte	2050.00	NULL	5
948	Pedregusto Mars	Rua Venus	3050.00	NULL	5
949	Laurinda Linda Lindeza	Rua Jupiter	5049.00	847	5

```
INSERT INTO empregado (matricula, nome, endereco, salario, mat supervisor, coddep)
VALUES (800, "Josivaldo Antunes Nunes", "Rua das flores", 4000, NULL, 1);
```


Sugestão de conteúdo para a tabela **dependente**.

id	matemp	nome	datan
1	949	Laurindinha Lindeza	2002-10-27
2	949	Laurindinho Lindeza	2002-10-27
3	947	Bikemotor Yamaha	2002-10-27
4	843	Pringles Doritos de Queijo	2002-10-28
5	848	Fanta Uva de Avila	2004-11-27
6	948	Setembrino Dorvalino	2012-12-26

```
INSERT INTO dependente (id, matemp, nome, datan)
VALUES (1, 949, "Laurindinha Lindeza", "2002-10-27");
```

```
INSERT INTO dependente (id, matemp, nome, datan)
VALUES (2, 949, "Laurindinho Lindeza", "2002-10-27");
```

Exercícios - Escreva a instrução SQL para:

1. Apresentar a listagem completa dos registros da tabela empregado;
2. Apresentar uma listagem dos nomes e salários dos empregados com salário maior do que 5.000;
3. Listar os nomes e os salários dos empregados em ordem alfabética, decrescente, de nome;
4. Listar os nomes dos empregados do departamento 5 ordenados pelo endereço;
5. Alterar para 2500,50 o salário dos empregados do departamento 2;
6. Aumentar em 20% o salário de todos os empregados;
7. Excluir todos os empregados do departamento 1;
8. Apresente a listagem dos dependentes que nasceram em 27/10/2002;
9. Apresente a listagem dos dependentes que nasceram após 27/10/2002;
10. Listar os empregados do departamento 5;
11. Listar empregados com salário até 5000,00;
12. Listar empregados com salário entre 1700,00 e 2800,00